COSC 123
Computer Creativity

Course Review

Dr. Ramon Lawrence
University of British Columbia Okanagan
ramon.lawrence@ubc.ca

Reading Data from the User
The Scanner Class

The `Scanner` class reads data entered by the user. Methods:

- `int nextInt()` - reads next integer
- `double nextDouble()` – reads next floating point number
- `String next()` – reads String (up to separator)
- `String nextLine()` – reads entire line as a String

To use must import `java.util.Scanner`.

```
import java.util.Scanner;
public class AddTwoNum
{  public static void main(String[] argv)
   {  // Code reads and adds two numbers
      Scanner sc = new Scanner(System.in);
      int num1 = sc.nextInt();
      int num2 = sc.nextInt();
      int result = num1+num2;
      System.out.println(num1+" + "+num2+" = "+result);
   }
}
```


Values, Variables, and Locations

A ***value*** is a data item that is manipulated by the computer.

A ***variable*** is the name that the programmer users to refer to a location in memory.

A ***location*** has an address in memory and stores a value.

IMPORTANT: The *value* at a given location in memory (named using a variable name) can change using initialization or assignment.

Compile vs. Run-time Errors

Question: A program is supposed to print the numbers from 1 to 10. It actually prints the numbers from 0 to 9. What type of error is it?

A) Compile-time error
B) Run-time error

Variables - Definitions

Question: Which of the following statements is correct?

A) The location of a variable may change during the program.

B) The name of a variable may change during the program.

C) The value of a variable may change during the program.

Assignment

Question: What are the values of `A` and `B` after this code?

```
int A, B;

A = 6;
B = 3;
A = 3 * B + A / B;
B = A + 5 * 3 * B;
```

A) `A = 6, B = 3`

B) `A = 11, B = 56`

C) `A = 5, B = 90`

Code Output

Question: What is the output of this code if user enters 3 and 4?

```
public class AddTwoNum
{  public static void main(String[] argv)
   {  // Code reads and adds two numbers
      Scanner sc = new Scanner(System.in);
      int num1 = sc.nextInt();
      int num2 = sc.nextInt();
      int result = num1+num2;
      System.out.println(num2+" - "+num1+" = "+result);
   }
}
```

A) 3 + 4 = 7
B) 4 + 3 = 7
C) 3 – 4 = 7
D) 4 – 3 = 7

Practice Question

1) Create a program to ask the user for two numbers, a operation to perform (+,-,/,*), and then do that operation.

Making Decisions

Decisions are used to allow the program to perform different actions in certain conditions.

To make a decision in a program we must do several things:

- 1) Determine the ***condition*** in which to make the decision.
- 2) Tell the computer what to do if the condition is true or false.
  - ⇨A decision always has a *Boolean value* or true/false answer.

The syntax for a decision uses the ***if*** statement:

```
if (age > 19)
   teenager=false;
```

OR

```
if (age > 19)
   teenager=false;
else
   teenager=true;
```


Making Decisions Block Syntax

Use the ***block syntax*** for denoting a multiple statement block. A block is started with a "**{**" and ended with a "**}**".

- All statements inside the brackets are grouped together.

Example:

```
if (age > 19)
{  teenager=false;
   hasLicense=true;
   ...
}
```

The ***dangling else problem*** occurs when a programmer mistakes an else clause to belong to a different if statement than it really does.

- Remember, blocks (brackets) determine which statements are grouped together, not indentation!

Nested Conditions and Decisions Boolean Expressions

A ***Boolean expression*** is a sequence of conditions combined using AND (**&&**), OR (**||**), and NOT (**!**).

- Allows you to test more complex conditions
- Group subexpressions using parentheses

Syntax: *(expr1)* **&&** *(expr2)* - expr1 AND expr2
(expr1) **||** *(expr2)* - expr1 OR expr2
!(expr1) - NOT expr1

Examples:

```
var b;
1) b = (x > 10) && !(x < 50);
2) b = (month == 1) || (month == 2) || (month == 3);
3) if (day == 28 && month == 2)
4) if !(num1 == 1 && num2 == 3)
5) b = ((10 > 5 || 5 > 10) && ((10>5 && 5>10));// False
```


Making Decisions Switch Statement

There may be cases where you want to compare a single integer value against many constant alternatives. Instead of using many if statements, you can use a ***switch*** statement.

- If there is no matching case, the default code is executed.
- Execution continues until the **break** statement. (Remember it!)
- Note: You can only use a switch statement if your cases are **integer numbers**. (Characters ('a', 'b',...,) are also numbers.)

Syntax:

```
switch (integer number)
{  case num1: statement break;
   case num2: statement break;
   ...
   default:   statement break;
}
```

Making Decisions (3)

Question: What is the output of this code?

```
int num=10;

if (num == 10)
{   System.out.print("big");
    System.out.println("small");
}
```

A) `big`

B) `small`

C) `bigsmall`

Making Decisions (4)

Question: What is the output of this code?

```
int num=10;

if (num >= 8)
   System.out.print("big");
   if (num == 10)
       System.out.print("ten");
else
    System.out.print("small");
```

A) `big`

B) `small`

C) `bigsmall`

D) `ten`

E) `bigten`

Switch Statement (2)

Question: What is the output of this code?

```
int num=2;

switch (num)
{  case 1: System.out.print("one");
   case 2: System.out.print("two");
   case 3: System.out.print("three"); break;
   default: System.out.print("other");
}
```

A) `onetwo`

B) `two`

C) `twothree`

D) `other`

E) `onetwothreeother`

Decision Practice Question

1) Write a program that reads a number *N*.

- If *N* has 1 digit, print 1 digit.
- If *N* has 2 digits, print 2 digits.
- Otherwise if *N* has 3 or more digits, print 3 or more digits.

The For Loop

The most common type of loop is the ***for loop***. Syntax:

```
for (<initialization>; <continuation>; <next iteration>)
{  <statement list>
}
```

Explanation:

- 1) initialization section - is executed once at the start of the loop
- 2) continuation section - is evaluated *before* every loop iteration to check for loop termination
- 3) next iteration section - is evaluated *after* every loop iteration to update the loop counter

Example:

```
int i;

for (i = 0; i < 5; i++)
{  System.out.println(i);     // Prints 0 to 4
}
```


For Loops

Question: What is the output of this code?

```
for (i=0; i < 6; i++)
   System.out.print(i);
```

A) nothing

B) error

C) The numbers 0, 1, 2, …, 5

D) The numbers 0, 1, 2, …, 6

For Loops

Question: What is the output of this code?

```
for (i=2; i < 20; i--)
   System.out.print(i);
```

A) nothing
B) infinite loop
C) The numbers 2, 3, 4, …, 19
D) The numbers 2, 3, 4, …, 20

Loops Practice Question

1) Write a program that reads a number *N* and calculate the sum of the numbers from 1 to *N*.

If that is too easy for you, calculate only the sum of the even numbers from 1 to *N*!

Java Object-Oriented Terminology

An ***object*** is an instance of a class that has its own properties and methods. Properties and methods define what the object is and what it can do. *Each object has its own area in memory.*

A ***class*** is a generic template (blueprint) for creating an object. All objects of a class have the same methods and properties (although the property values can be different).

A ***property*** (or ***instance variable***) is an attribute of an object.

A ***method*** is a set of statements that performs an action. *A method works on an implicit object and may have parameters.*

A ***parameter*** is data passed into a method for it to use.

Access Specifiers Public and Private

One of the features of object-oriented programming is that not all parts of a program have access to all the data and methods.

Each class, method, and variable needs to have one of the four ***access specifiers*** defined that indicate which other objects and methods in your program have access to it. Four types:

- `public` – Accessible by all code (everyone, the public)
- `private` – Only accessible by methods in the class.
- `protected` – Only accessible by methods in the class or classes derived from this class by inheritance.
- default – If nothing is specified, assume package access where all methods in same package can access it.

Class Example `BankAccount` Class

```
public class BankAccount
{
   private double balance;

   public void deposit(double amount)
   {  balance = balance + amount; }

   public void withdraw(double amount)
   {  balance = balance - amount; }

   public double getBalance()
   {  return balance; }
}
```

The `BankAccount` class is used for describing bank accounts.

- The methods defined in the `BankAccount` class are `deposit`, `withdraw`, and `getBalance`.
- The current balance in the account is `private`, so it can only be changed by calling the methods.

Class Practice Questions

1) Write the `setBalance` method for the `BankAccount` class.

2) Add an instance variable called name for the name of the owner of the `BankAccount`. Add get/set methods for this instance variable.

3) Add a default constructor (no parameters) and an overloaded constructor that accepts balance and name as parameters.

Creating and Using Objects

A class is just a blue-print for creating objects.

- By itself, a class performs no work or stores no data.

For a class to be useful, we must create objects of the class.

- Each object created is called an ***object instance***.

To create an object, we use the **`new`** method.

When an object is created using the `new` method:

- Java allocates space for the object in memory.
- The ***constructor*** for the object is called to initialize its contents.
- Java returns a pointer to where the object is stored in memory which we will call an ***object reference***.

Objects and Object References

Question: How many object references are in this code?

```
BankAccount savings, checking;
BankAccount myAcct, myAcct2;

savings = new BankAccount();
myAcct = savings;
checking = new BankAccount();
```

A) 1
B) 2
C) 3
D) 4

Objects and Object References

Question: How much money is in the account referenced by the `myAcct2` object reference?

```
BankAccount savings, checking;
BankAccount myAcct, myAcct2;

savings = new BankAccount(50);
myAcct = savings;
savings = null;
checking = new BankAccount(100);
savings = checking;
myAcct2 = myAcct;
```

A) unknown
B) 50
C) 100
D) undefined

Variable Scope
Scope of Variable Types

The scope of variables depends directly on their type:

- 1) ***Instance variables*** - are created when an object instance is created using the `new` method. Instance variables are defined as long as there is at least one reference to the object in your program which is still in scope.
- 2) ***Static variables*** - are created when the class they are defined in is first loaded and are defined until the class is unloaded.
  - ⇨This means static variables are around for the duration of your program.
- 3) ***Local variables*** - are created when the program enters the block in which they are defined and destroyed when the program exits that block.
  - ⇨A variable defined in brackets ("{","}") is accessible anywhere within the block including nested blocks.
- 4) ***Parameter variables*** - are created when a method is first called and are destroyed when a method returns.

Variable Scope
Practice Questions

With this code explain the lifetime and scope of all variables.

```
public class VariableScope
{  public static void main(String[] args)
   {  double amount = 25;
      BankAccount acct = new BankAccount(200);
      for (int i=1; i <= 3; i++)
         acct.deposit(amount);
      System.out.println(acct.getBalance());  // 125.0
   }

   private void doNothing(double a)
   {  int i = 5; return;   }

   public static final int MYNUM = 25;
}
```


Variable Scope
Practice Questions (2)

```
class BankAccount
{  public void deposit(double amount)
   {  if (amount <= balance)
      {  double newBalance = balance - amount;
         balance = newBalance;
      }
      double balance = 50;
   }
   public double getBalance()
   {  return balance; }

   public BankAccount(double b)
   {  balance = b; lastAccountNum++;
      accountNum = lastAccountNum;
   }

   private double balance;
   private int accountNum;
   private static int lastAccountNum = 0;
}
```

Inheritance Overview

Inheritance is a mechanism for enhancing and extending existing, working classes.

- ⇨ In real life, you inherit some of the properties from your parents when you are born. However, you also have unique properties specific to you.
- ⇨ In Java, a class that extends another class inherits some of its properties (methods, instance variables) and can also define properties of its own.

`Extends` is the key word used to indicate when one class is related to another by inheritance.

Syntax: `class` *`subclass`* **`extends`** *`superclass`*

- The ***superclass*** is the existing, parent class.
- The ***subclass*** is the new class which contains the functionality of the superclass plus new variables and methods.

Inheritance Question

1) Create a `CheckingAccount` class which inherits from `BankAccount`. The `CheckingAccount` class:

- inherits `getBalance()` from `BankAccount`
- overrides `deposit()` and `withdraw()` from `BankAccount`, so it can keep track of the number of transactions (`transactionCount`)
- defines a method `deductFees()` which withdraws $1 for each transaction (`transactionCount`) then resets the # of transactions

Arrays

An ***array*** is a collection of data items of the same type.

An array reference is denoted using the open and close square brackets "`[]`" during declaration.

- ⇨ You can have an array of any data type including the base types (`int`, `double`, `String`) and object-types (`BankAccount`).

- Examples:

```
int[] myArray;
String[] strings;
BankAccount[] accounts;
```

Similar to an object, when you declare an array you are creating a ***reference*** to an array. Until you actually create the space for the array using **`new`**, no array exists in memory.

- `String[] strings = new String[10];`

Arrays

Question: What is the size of this array?

```
int[] myArray = new int[10];
```

A) error
B) 10
C) 9
D) 11

Arrays

Question: What are the contents of this array?

```
int[] myArray = new int[4];
myArray[0] = 1;
myArray[3] = 2;
myArray[2] = 3;
myArray[0] = 4;
```

A) error
B) 0, 1, 2, 3
C) 1, 2, 3, 4
D) 4, 0, 3, 2

ArrayLists

An `ArrayList` implements a *resizable* array of objects.

- ⇨ Base types such as int are not objects. Use wrapper class Integer.

Create an `ArrayList` by:

```
ArrayList names = new ArrayList(); // Size 10 (default)
ArrayList accounts = new ArrayList(5); // Size of 5
```

Add element to an `ArrayList` by:

```
names.add("Joe");       // Add to end of list
names.add(2,"Steve");   // Add at index 2 and shift down
```

Remove element from an `ArrayList` by:

```
names.remove(2);        // Remove index 2 and shift up
```

ArrayLists (2)

Get number of items in list by:

```
int count = names.size();
```

Get element at an index from an `ArrayList` by:

```
String n = names.get(2);   // Get item at index 2
```

Set element at an index in an `ArrayList` by:

```
names.set(2,"Fred");       // Put Fred at index 2
```

A simple way to traverse an `ArrayList` is using a for loop:

```
for (int i=0; i < names.size(); i++)
{  String s = (String) names.get(i);
   System.out.println(s);
}
```


ArrayList

Question: What is the value of `st`?

```
ArrayList a = new ArrayList();
a.add("Fred");
a.add(1,"Joe");
a.add(1,"Steve");
a.remove(0);
String st = (String) a.get(0);
```

A) Fred
B) Joe
C) Steve
D) error

ArrayList Practice Question

1) Write a method that takes an `ArrayList` as a parameter and returns an `ArrayList` with all the items in reverse order.

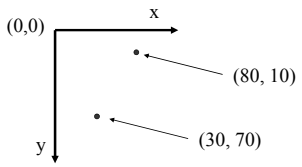
For example, if the list was `{Joe, Fred, Smith}` then after reverse the list is `{Smith, Fred, Joe}`.

The Coordinate System

Drawing on the screen is done by specifying coordinates which refer to a location on the screen.

- The ***origin*** is the upper-left hand corner of the screen.
- The x coordinate gets bigger as we move to the right.
- The y coordinate gets bigger as we move down.

Diagram:



Drawing Methods

- 1) Ellipse:
  - Ellipse2D.Double egg = new Ellipse2D.Double(topx, topy, width, height);
  - Ellipse2D.Double egg = new Ellipse2D.Double(5, 10, 15, 20);
- 2) Rectangle:
  - Rectangle box = new Rectangle(topx, topy, width, height);
  - Rectangle box = new Rectangle(10, 10, 20, 30);
- 3) Line:
  - Line2D.Double = new Line2D.Double(x1,y1, x2, y2);
- 4) Point:
  - Point2D.Double = new Point2D.Double(x,y);

You can also fill a shape with a color using the `fill` method:

- g2.fill(box);
- g2.fill(egg);

Drawing Methods (2)

- Change colors:
  - `g2.setColor(Color.orange);`
- Draw a string
  - `g2.drawString("Hello", 50, 100); // message, x, y`

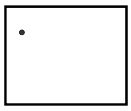
Java Swing Components

The Java Swing package contains the user interface components that we will use in our graphical applications.

Component
JButton
ButtonGroup
Check box
Combo box
JFrame
JLabel
JPanel
Radio button
Text field
JMenuBar
JMenu
JMenuItem

Coordinates

Question: Select from the coordinates below the pair that best describes this point's location. Assume box is 100 by 100.



A) (10,80)
B) (80,10)
C) (10,20)
D) (20,10)

Events and Event Handling Overview

An ***event*** is a notification to your program that something has occurred.

- For graphical events (mouse click, data entry), the Java window manager notifies your program that an event occurred.
  - There are different ***kinds*** of events such as keyboard events, mouse click events, mouse movement events, etc.

An ***event handler*** or ***listener*** is part of your program that is responsible for "listening" for event notifications and handling them properly.

- An event listener often only listens for certain types of events.

An ***event source*** is the user interface component that generated the event.

- A button, a window, and scrollbars are all event sources.

Mouse Event Example

Handling mouse click events requires three classes:

- 1) **The event class** - that stores information about the event.
  - For mouse clicks, this class is `MouseEvent`.
  - The `MouseEvent` class has methods `getX()` and `getY()` that indicate the position of the mouse at the time the event was generated.
- 2) **The listener class** - allows your program to detect events. Building your own listener class requires implementing a pre-defined interface.
  - For mouse clicks, the listener interface is `MouseListener`.
  - An ***event listener*** is a class that implements all methods of an event interface.
  - An ***event adapter*** extends a class and only requires you implement method for the events that you are interested in.
- 3) **The event source** - is the component in your GUI that generated the event.

Mouse Event Example Code

```
import java.applet.Applet;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;

public class MouseSpyApplet extends Applet
{  public MouseSpyApplet()
   {   MouseSpy listener = new MouseSpy();
       addMouseListener(listener);
   }
}

class MouseSpy implements MouseListener
{   public void mouseClicked(MouseEvent event)
    {  System.out.println("Mouse clicked. x = " + event.getX() + " y = " + event.getY());
    }
    public void mouseEntered(MouseEvent event)
    {  System.out.println("Mouse entered. x = " + event.getX() + " y = " + event.getY());
    }
    public void mouseExited(MouseEvent event)
    {  System.out.println("Mouse exited. x = " + event.getX() + " y = " + event.getY());}
    public void mousePressed(MouseEvent event)
    {  System.out.println("Mouse pressed. x = " + event.getX() + " y = " + event.getY());
    }
    public void mouseReleased(MouseEvent event)
    {  System.out.println("Mouse released. x = " + event.getX() + " y = " + event.getY());
    }
}
```


Exception Handling

An ***exception*** is an error situation that must be handled or the program will fail. ***Exception handling*** is how your program deals with exceptions when they occur.

Two ways of handling exceptions:

- 1) *Handle* them inside the method using a `try-catch-finally` block.
- 2) *Throw* the exception to the method that called it and force that method to handle it.

Two types of exceptions:

- ***Checked exceptions*** are exceptions that you must tell the compiler how your code is handling them. (e.g. `IOException`)
  - A checked exception *must* be either caught or thrown.
- ***Unchecked exceptions*** are exceptions that the compiler does not force your program to handle.

The try-catch-finally Statement

The ***try-catch-finally statement*** identifies a block of statements that may throw an exception and provides code to handle exceptions if they occur.

Three components:

- ***try block*** - has statements to execute that may cause exceptions. Each statement is executed one at a time. If an exception occurs, jump out of try block to a catch clause. If no exception, go to finally clause (if it exists).
- ***catch block*** – handles a particular kind of exception and has code that performs the desired action if it occurs. Only one catch clause is every executed and are not executed if an exception does not occur.
- ***finally block*** – code that is always executed regardless if all statements completed successfully or an exception occurred

Exceptions

Question: TRUE or FALSE: An uncaught exception may be passed through several methods before the program crashes.

A) TRUE

B) FALSE

Exceptions

Question: What does this code output if the user enters "32"?

```
try
{  Scanner sc = new Scanner(System.in);
   System.out.print("Enter a number: ");
   int num = sc.nextInt();
   System.out.print(num+" ");
}
catch (InputMismatchException e)
{  System.out.print("Input was not a number. ");
}
finally
{  System.out.print("HELLO!");
}
```

A) nothing

B) `32`

C) `Input was not a number.`

D) `32 HELLO!`

Read Text File with `Scanner`

```
Scanner sc = null;
try
{  sc = new Scanner(new File("MyFile.txt"));
   while (sc.hasNextLine())
   {  String st = sc.nextLine();
      System.out.println(st);
   }
}
catch (FileNotFoundException e)
{  System.out.println("Did not find input file: "+e); }
finally
{  if (sc != null)
      sc.close();
}

Note: The Scanner class handles some exceptions for you
and makes it easier to read numbers and other types that
are not Strings.  It should be the one used.
```


Streams and Exceptions Practice Question

1) Write a program that opens up the file "test.txt" that contains numbers and computes a sum where every odd number is added and every even number is subtracted.

Putting it all together...

Computer programming is the art and science of solving problems on the computer.

- As you have seen, there are many different ways to approach and solve the same problem.
  - ⇨Each technique may have different benefits and performance.

Computer science is about learning how to make the correct and most efficient decisions on how to solve problems.

- ⇨Anyone can program on a computer, but computer scientists know why they are programming a solution and what they are doing.

The most exciting aspect of programming is the satisfaction of building a program to solve a problem.

- Whether it is a simple algorithm or a program that runs a nuclear power plant.
  - ⇨You do not quite have the skills or experience to solve the large problems, but you can solve smaller problems and appreciate the difficulty inherent in solving the larger ones.